

IME I PREZIME: \_\_\_\_\_ Ak. god. 2022./2023.

JMBAG: \_\_\_\_\_

## 2. domaća zadaća iz Formalne verifikacije programske potpore

### Java PathFinder

Najprije je potrebno instalirati sustav Java PathFinder prema uputama u datoteci "Java\_PathFinder\_instalacija.pdf"

Svrha 1. dijela 2. domaće zadaće je upoznavanje s projektom `jpf-core` i pokretanje provjere modela jednostavnih primjera programa. U 2. dijelu domaće zadaće uvode se dodatni razredi "slušači" koji nadograđuju osnovnu funkcionalnost projekta `jpf-core`. 3. dio domaće zadaće pokriva izvođenje provjere modela nad primjerima iz projekta `jpf-aprop`, koji je dodatni projekt koji se može koristiti za provjeru različitih anotacija u programu. Konačno, u 4. dijelu domaće zadaće studenti će napraviti svoj projekt od početka, uključiti korištenje projekata `jpf-core` i `jpf-aprop` te provjeravati model zadanog programa uz potrebne izmjene.

#### 1. dio - Provjera modela za jednostavne primjere u projektu `jpf-core`

Proučite strukturu projekta `jpf-core`. Primjeri nad kojima će se raditi provjera modela nalaze se u paketu `src/examples`. Osim ako nije drugačije zadano, programi se u NetBeansu mogu pokrenuti desnim klikom na odgovarajuću `*.jpf` datoteku i odabirom opcije "Verify..."

Alternativno, ako niste uspjeli podesiti *plugin* "Verify..." da radi (vidjeti instalacijske upute), moguće je pokrenuti program unoseći puni put u komandnoj liniji koji je obično ovakvog oblika:

```
java -jar C:\Users\Korisnik\projects\jpf\jpf-core\build\RunJPF.jar
+shell.port=4242 C:\Users\Korisnik\Documents\NetBeansProjects\jpf-
core\src\examples\HelloWorld.jpf
```

**1.1.** Otvorite konfiguracijsku datoteku projekta `jpf-core` koja se naziva `jpf.properties` i koja se nalazi u korijenskom direktoriju projekta. Do datoteke se može doći putem kartice "Files" koja se nalazi pokraj kartice "Projects" u NetBeansu ili putem Windows Explorera.

Koja se *defaultna* strategija koristi za pretragu prostora stanja u JPF-u? Koja se standardna svojstva provjeravaju prilikom pretrage korištenjem odgovarajućih slušača? Osim naziva strategije i svojstava, navedite i puna kvalificirajuća imena razreda u projektu `jpf-core` koji za to služe.

**1.2.** Proučite najjednostavniji primjer aplikacije `HelloWorld.java` (u paketu `examples`). Što se ispisuje pokretanjem provjere modela tog programa u dijelu nakon `search started`? Koje su pogreške dojavljene?

**1.3.** Proučite primjer ograničenog međuspremnika `BoundedBuffer.java`. Navedite koji su sudionici u ovom primjeru.

**1.4.** Kojim mehanizmom u Javi su ostvareni ti sudionici? Koje metode koriste koji sudionici?

**1.5.** Pokrenite aplikaciju ograničenog međuspremnika. Koje svojstvo je narušeno izvođenjem ovog programa koristeći argumente navedene u konfiguracijskoj datoteci (2,4,1)? Što se dogodilo s pojedinim sudionicima? Kolika je bila veličina međuspremnika u ovom slučaju?

**1.6.** Pokrenite istu aplikaciju, samo s argumentima (4,1,1). Kakva je sad situacija? (Napomena: NetBeans će vas možda gnjaviti da ne može spremiti izmjene u datoteci `*.jpf` jer da je datoteka otvorena vjerojatno samo za čitanje. Obično spremanje promjena ipak uspješno prođe nakon što prođe neko vrijeme (manje od minute), no u slučaju da ne prođe, možete napraviti `Save As...` i pohraniti datoteku pod drugim imenom)

**1.7.** Opišite ukratko što rade i nad čime se pokreću Javine metode `wait()` i `notify()`.

**1.8.** Obrazložite ukratko (i precizno) zašto dolazi do narušavanja svojstva u ovom primjeru.

**1.9.** Proučite primjer `Rand.java` i pridruženu konfiguracijsku datoteku `Rand.jpf`. Što specificira konfiguracijska naredba `cg.enumerate_random = true` i zašto je ona bitna za ovaj problem?

**1.10.** Pokrenite aplikaciju za verifikaciju. Koje svojstvo je ovdje narušeno? Što je programer ovog ili ovome sličnog koda previdio? Kako se mogao unaprijed osigurati da se cijeli sustav ne sruši? Koje su konkretne vrijednosti varijabli `a` i `b` srušile program?

**1.11.** Sad izbrišite specifikaciju `cg.enumerate_random = true` iz konfiguracijske datoteke te pokrenite aplikaciju. Što se sad dogodilo? Objasnite.

## **2. dio - Provjera modela u projektu `jpf-core` korištenjem dodatnih slušača**

**2.1.** Proučite primjer `Racer.java` i konfiguracijsku datoteku `Racer.jpf`. Napravite kopiju konfiguracijske datoteke koju ćete nazvati `Racer_2.jpf` i u kojoj ćete izbrisati liniju `listener=gov.nasa.jpf.listener.PreciseRaceDetector`. Pokrenite aplikaciju za verifikaciju putem `Racer.jpf` i zatim putem `Racer_2.jpf`. Koje svojstvo je narušeno u slučaju `Racer_2.jpf`, a što piše pod error 1 u slučaju `Racer.jpf`?

**2.2.** Opišite zašto može doći do problema prilikom izvođenja ovog primjera. Može li instanca dretve `t` pristupiti liniji koda `int c = 420 / racer.d;` ?

**2.3.** Otvorite kod razreda `gov.nasa.jpf.listener.PreciseRaceDetector`. Ukratko pojasnite (na temelju komentara razreda) koja je ideja kod implementacije detektora utrke za resursom. Također navedite koji Adapter nasljeđuje ovaj slušač i koje metode nadjačava.

**2.4.** Proučite kod primjera `NumericValueCheck.java` i konfiguracijsku datoteku `NumericValueCheck.jpf`. Zatim pokrenite aplikaciju. Koju grešku je dojavio JPF?

**2.5.** Primijetite na koji način je specificirano u konfiguracijskoj datoteci na koju varijablu i na koji način se odnosi provjeravanje raspona numeričkih vrijednosti. Pogledajte sad kod odgovarajućeg slušača koji implementira provjeravanje raspona vrijednosti. Koje su dvije mogućnosti rada tog slušača (na koje dijelove nekog razreda se može primijeniti)? Navedite i sintaksu tih provjera.

**2.6.** Proučite kod primjera `TestExample.java` i konfiguracijsku datoteku `TestExample-coverage.jpf`. Korištenjem slušača `CoverageAnalyzer` omogućena je analiza pokrivanja koda. Pokrenite aplikaciju i promotrite rezultatnu tablicu koju je ispisao `CoverageAnalyzer`. Koji razred je bio bolje pokriven s ispitnim primjerima u metodi `main`? Koje sučelje je morao implementirati ovaj slušač kako bi izmijenio izgled ispisa izvještaja?

**2.7.** S obzirom na rezultate ispisa i dani kod, koja bi to bila metoda `<init>()` koja piše u tablici?

**2.8.** Dodajte u konfiguracijsku datoteku `TestExample-coverage.jpf` pod razrede koje treba uključiti za provjeru dodatno i sam razred `TestExample`, uz postojeće razrede `T1` i `T2`. Spremite datoteku i pokrenite aplikaciju za verifikaciju. Proučite rezultat. Iz samog koda, očito je da će se proći kroz sve linije metode `main`. Koje naredbe (linije koda) analizador pokrivanja preskače kad izvještava da je prošao kroz samo 3/8 linija koda metode `main` (osma linija koda uključuje implicitni `return;`)? Kojom specifikacijom bi isključili ispisivanje provjeravanja pokrivenosti grana u izlaznoj tablici?

### 3. dio - Provjera modela u projektu jpf-aprop

Instalirajte projekt (paket) jpf-aprop koji služi za provjeru specifičnih svojstava programa pisanih u Javi koja su zadana u kodu u obliku anotacija (oznaka @). Projekt možete naći u zip obliku na Moodleu predmeta FVPP, u materijalima za 2. DZ. Projekt raspakirajte u isti direktorij gdje se nalazi i projekt jpf-core (npr. NetBeansProjects).

U NetBeansu otvorite novi projekt: odaberite "File" -> "New Project" -> "Java" -> "Java Free-Form Project". Pod "Location" odaberite direktorij gdje se nalazi jpf-aprop. Ostale sve rubrike bi se trebale popuniti automatski. Nastavite dalje. U koraku "Source Package Folder" pod "Source level" izaberite JDK 1.8. Zatim odaberite "Finish". jpf-aprop bi se trebao naći u otvorenim projektima. Dodajte u datoteku site.properties, koju ste ranije stvorili u postupku instalacije, sljedeće retke na kraj datoteke i pohranite promjene.

```
# annotation properties extension
jpf-aprop = ${jpf.home}/jpf-aprop
extensions+=", ${jpf-aprop}"
```

Sada pokrenite skriptu build.xml koja se nalazi u korijenskom direktoriju projekta jpf-aprop (desni klik pa "Run Target" -> "build"). Prevođenje datoteka i izgradnja tri .jar datoteke trebalo bi proći bez pogrešaka.

**3.1.** Proučite datoteku jpf.properties projekta jpf-aprop. Koja se standardna svojstva provjeravaju prilikom pretrage kad se koristi projekt jpf-aprop? Gdje je to uopće definirano? Također, navedite put do direktorija s međukodom razreda koji se kao primjeri provjeravaju uz pomoć jpf-aprop. Koje svojstvo pokazuje taj put u datoteci jpf.properties?

**3.2.** Proučite jednostavni primjer ConstViolation.java i pridruženu konfiguracijsku datoteku ConstViolation.jpf te odgovarajućeg slušača. Opišite što se događa u kodu razreda ConstViolation.java. Koja metoda je označena s @Const anotacijom i što to točno znači?

**3.3.** Pokrenite tu aplikaciju za verifikaciju. Navedite pogrešku koja je dojavljena.

**3.4.** Proučite primjer `ContractViolation.java` i pridruženu konfiguracijsku datoteku `ContractViolation.jpf`. Navedite koje sve slušaće koristi ovaj program (puna kvalificirajuća imena).

**3.5.** Pronađite u strukturi projekta odgovarajućeg slušaća u kojem se provjerava svojstvo `nonshared.throw_on_cycle`. U kodu pronadite i napišite koju bi vrstu iznimke bacio JPF ako bi dretva bila uhvaćena u ciklusu nad objektom koji nije predviđen za višedretvenost.

**3.6.** Što znače anotacije `@Requires`, `@Invariant` i `@Ensures` u kodu programa `ContractViolation.java`? Koju vrstu dobrog oblikovanja programske potpore ostvaruju ove anotacije?

**3.7.** Pokrenite aplikaciju za verifikaciju. Koja anotacija je narušena? Napišite pogrešku koja je dojavljena.

**3.8.** Pažljivo proučite dojavljenu pogrešku. Koja metoda kojeg točno razreda je izazvala narušavanje ugovora?

**3.9.** Promijenite ugovore dviju metoda tako da obje ispituju uvjet (`Result>=0`). Spremite izmijenjenu datoteku `ContractViolation.java`. Pokrenite ponovno skriptu `build.xml`. Kad se izmijenjeni primjer preveo, ponovno ga pokrenite. Kakva je sad situacija?

#### **4. dio - Provjera modela vlastitog projekta**

**4.1.** U NetBeansu napravite novi projekt ("File" -> "New Project" -> "Java Application") koji ćete nazvati `JavaFV`. Napravite ga bez razreda `JavaFV` s metodom `main`. Zatim desnim klikom na `Source Packages` napravite novi paket pod nazivom `fv`, a onda desnim klikom na paket `fv` napravite novi razred pod imenom `Verifikacija.java` i statičkom metodom `main` (unutar razreda napišite `public static void main(String[] args) {}`).

Napravite unutar istoga paketa novu datoteku (desni klik na `fv`, pa "New" -> "Other..." -> "Other" -> "Empty File" i nazovite ga `Verifikacija.jpf`. U tu datoteku dodajte zasad samo jedan redak kojim ćete omogućiti pokretanje razreda `Verifikacija.java` iz paketa `fv` i spremite ju. Kako izgleda taj redak?

U korijenskom direktoriju projekta `JavaFV` zatim napravite datoteku `jpf.properties` jednostavnog sadržaja:

```
JavaFV = ${config_path}
```

```
JavaFV.classpath=\n${JavaFV}/build/classes
```

```
JavaFV.sourcepath=\n  ${JavaFV}/src/fv
```

Na kartici "Projects" kliknite desnim klikom na vaš projekt `JavaFV` i odaberite "Clean and build".



Nakon što se projekt izgradio, provjerite da se međukod `Verifikacija.class` nalazi pod direktorijem `build/classes/fv`. Probajte pokrenuti verifikaciju koja bi trebala proći bez pogrešaka.

Objasnite zašto je redak `JavaFV.classpath=\`  
`${JavaFV}/build/classes`  
nužno navesti u datoteci `jpf.properties`?

**4.2.** Sada izmijenite sadržaj datoteke `Verifikacija.java` tako da sadrži kod koji se nalazi u istoimenoj datoteci koja se nalazi u repozitoriju kolegija FVPP na Moodleu (pod 2. DZ). Također, izmijenit ćete sadržaj datoteke `Verifikacija.jpf` tako da sadrži specifikacije prema istoimenoj datoteci koja se nalazi u repozitoriju kolegija. Nakon kopiranja koda i specifikacija spremite datoteke, no nećete moći prevesti datoteku `Verifikacija.java` budući da sadrži `importe` koji su nepoznati projektu `JavaFV`.

**4.3.** Potrebno je uključiti izgrađene knjižnice (`.jar`) od `jpf-core` i `jpf-aprop` u projekt `JavaFV` kako bi se kod razreda `Verifikacija.java` mogao prevesti. To se radi tako da odaberete `JavaFV` pa desni klik, a zatim "Properties" -> "Libraries" -> "Add JAR/Folder". Pronađite u direktoriju `jpf-core` -> `build` -> `jpf.jar`, `jpf-classes.jar` i `jpf-annotations.jar` te ih dodajte. Ostale knjižnice nisu bitne. Od projekta `jpf-aprop` potrebno je dodati samo knjižnicu `jpf-aprop-annotations.jar`. Odaberite "Ok". Pogreške bi sada trebale nestati. Sad prevedite `Verifikacija.java` (desni klik -> "Compile File").

**4.4.** Pokrenite aplikaciju za verifikaciju. Koja pogreška vam je dojavljena? Objasnite zašto je došlo do te pogreške s obzirom na konfiguracijsku datoteku i zadani kod.

**4.5.** U konfiguracijsku datoteku dodajte ovaj redak na kraj:

```
search.multiple_errors = true
```

Ovime se prolazi svim putevima izvođenja kroz program i dojavljuje se za svaki put izvođenja prva pogreška na koju se naletilo. Pokrenite aplikaciju za verifikaciju. Koja je razlika između prethodnog ispisa pogrešaka i sadašnjega?

**4.6.** Uklonite redak `search.multiple_errors = true` te zakomentirajte redak u kodu koji smatrate odgovornim za dojavu pogreške iz zadatka 4.4. Prevedite kod i pokrenite ponovno aplikaciju za verifikaciju. Koja se sada pogreška pojavila, na kojem retku koda i zašto je prijavljena?

**4.7.** Zakomentirajte redak u kodu koji smatrate odgovornim za dojavu ove vrste pogreške. Prevedite kod i pokrenite ponovno aplikaciju za verifikaciju. Koja je sada pogreška dojavljena? Objasnite zašto se ova pogreška dogodila.

**4.8.** Izmijenite naredbu (`assert`) u kodu tako da više ne dolazi do ove vrste pogreške. Prevedite kod i provjerite da se pogreška zaista više ne događa. Koji ste broj trebali navesti kao uvjet u naredbi (`assert`) da ne dođe do pogreške?

**4.9.** Provjerite metode razreda `gov.nasa.jpfnm.Verify`. Postoje li metode `getDouble()` i `getInt()` bez argumenata? Objasnite. Objasnite na primjeru što su to generatori izbora i koja je namjena navođenja heuristika pri korištenju generatora izbora. Koja se heuristika koristila u primjeru `Verifikacija.java`?